01-T2 - Minecraft Game Mode

Final Report

CS 4850 – Section [01/03/04] – Spring 2025, April 28th

Ashley Ahn, Matthew Elledge, AnnaGrace Gwee,

Bryan Nguyen, & Logan Slicker

Professor Sharon Perry

Website & GitHub: https://github.com/KsuBlocksTD/BlocksTD1.0

Lines of Code: 9375

Number of Project Components & Tools: 10

Estimated Hours: 153 | Total Hours: 326

Status: Project is 100% Complete

Table of Contents

Introduction

Our project, KSUBlocks Tower Defense, is a Minecraft Plugin designed to create a game mode in the Tower Defense genre. With the aim of creating a fun and unique game for the students in the KSU Minecraft Server, the project is fully customizable, allowing for easy maintenance, expansion, and balance. Our project is designed in Java, utilizing IntelliJ and Paper API and will be deployed on the KSU Minecraft Server upon completion.

Requirements

Requirements are three-part: Milestones, Functional and Non-Functional. Milestones will detail all items necessitated by the Industry Sponsor. Functional will tackle the necessary code implementation for game functionality, while non-functional will detail the information needed for the design and development elements of the game.

Milestones

Milestone 1

The first milestone was dedicated to the following tasks:

- Draft the game design document (GDD) for our plugin (Viewable on our GitHub)
    - The command structure and permissions were defined for our game, essentially coding the basis of the project with pen and paper.
    - Libraries that would be used for both the game and to work with the server were considered and researched during this time.
    - Game mechanics, rules, and considerations of multiple players were drafted during this time.
    - Plans were created on how we would handle the world, in which we settled upon maps that the players could load into similar to a campaign.
    - General information was also supplied regarding the game genre and how to explain it through laymen terms and a high technical level.
- GitHub Repository & IDEs are to be set up to begin coding after GDD approval.

Milestone 2

The second milestone was dedicated to the following tasks:

- Verify asynchronous compatible processes are handled off the main thread.

    - This was accomplished through a function dedicated to testing this alongside ensuring code was well documented and comments are supplied within the code for detailed explanation.
- The majority of the game code should be complete with a basic prototype
    - This was showcased in a prototype demo showing tower functionality and incoming rounds of mobs towards an end path
- Maps are to be designed and completed during this time
    - In this time, two functional maps and two rough draft maps were released.

Milestone 3

The third and final milestone were dedicated to the following tasks:

- The game is to be polished and made user friendly
    - User Interfaces were given to map selection, tower purchases and player upgrades, and tower upgrades.
    - Towers were adjusted to be placed anywhere and given skins for differentiation.
    - All four maps were completed and polished with new visuals.
    - A hub world was created with a leaderboard showcasing player stats.
- Bug Testing and Playtesting is to be completed
    - After software testing, the game was fully functional with no major known bugs, with a complete playable demo present in C-Day having no issues.
- Documentation on set-up, commands, permissions, and configs is required
    - This was completed on our GitHub pages: We supply all artifacts throughout the semester on the home page alongside full documentation for players, admins, and the plans of the team on our GitHub Wiki pages.

Functional Requirements

Map Selection

The game mode needs to be able to handle player map selection and work with all implemented maps at launch (4). The players should be able to select a map and have all functions work accordingly. All maps should present differences. This has been accomplished by allowing the players to spawn into the hub zone and select any of the four maps present on launch through a compass GUI in their inventory. Maps are loaded as needed and players are sent back to the hub upon a game completion or failure.

Round Management

The game mode should be able to create the game state and allow players to ready up and start the next round when they are prepared. Round management should also handle

income, incoming mobs, and ensure game rules and mechanics are followed. To handle this, functions regarding the round correspond to other elements of the game, notably the player starting the game alongside functionality of the gameplay itself (enemies, mobs, towers, items, and so on). These functions are also asynchronous to maintain stable and consistent server performance.

Mob Management

Mobs should properly follow the laid-out path towards the objective, interact accordingly with other functions (income, loss, health, towers), and present their unique attributes properly. These mobs mostly have self-contained functions that can be called when game logic necessitated it alongside utilizing economy game logic for communication.

Tower Management

Towers should be able to be built with enough income, properly have upgraded stats upon prompt, and show the user a GUI that presents purchase, upgrades, and stats. Towers should also handle income management to the player and accurately damage mobs. All towers are built off of one main tower function and call the tower type function when placed to maintain asynchronous behavior.

Player Management

Players should be able to attack mobs, build towers, upgrade their own attributes (both pertaining to their weapons and their player's effects), and be provided with the necessary items and functions to play the game. Players handle one major upgrade function to allow access to other items such as towers and buffs and correspond to the necessary economy functions for tower and upgrade behavior. The player also has a function to generate the tower spawn eggs utilizing Minecraft's vanilla features. Players are also given a leaderboard that is updated with their current health, coins, and general game information.

Non-Functional Requirements

Capacity

The game should be able to run on any Java 1.21.4 Paper server. It should be lightweight enough to be able to handle higher round difficulties and mob spawning, as well as be able to handle multiple people in each game performing multiple action simultaneously. This is accomplished through asynchronous behavior and further tested with a function solely to ensure asynchronous threads are working. The game also utilizes the base game's architecture to call necessary data and functions to minimize extra workload.

Usability

The game should be easy to implement for the developers and any server's staff, while being easy to understand for both administrators and players. Through the usage of

documentation on the GitHub Wiki alongside a downloadable package, staff and future developers can easily install the game through the GitHub page. The Repo hosting the game is also open source, allowing any developer to modify the game as needed.

Modifiability

The game should allow for full customization through configuration, ease of new implementations, and changes for balances as needed. As documented on the GitHub Wiki, future developers and staff can easily reference the material as needed, whether it be through rebuilding the plugin or creating new maps. Furthermore, the config.yml file is fully documented and self-explanatory while allowing for nearly every aspect of the game to be changed and easily reloaded on the server.

Analysis & Design

Tools & Dependencies

Plugins

- Parties – Creates the party system for multiplayer
- PlaceholderAPI – Displays plugin information uniformly for servers
- SimpleScore – Creates the leaderboard system (optional)

APIs

- Paper – The platform for many modded Minecraft servers
- Parties
- PlaceholderAPI

IntelliJ MC Dev Plugin

- The basis for many Minecraft Plugins, including KSUBlocks

General Constraints

Asynchronous

- Using async tasks for saving/keeping less important data or time sensitive data will be vital to ensure the main processing thread is kept open and unblocked. Server performance without latency issues is key.

Configurable

The game should have many of its values modifiable through the config file

- The game should be easy to maintain outside of the project's original team
  - Code should be well documented and commented
- The game should be easy to expand upon
  - Instructions on how to make new apps
  - Easy to use functions for future additions to the game

Performance

- The game should run smoothly to ensure no issues during gameplay and no issues while hosting on the KSU MC server alongside other game modes. Code should be efficient and well documented to reflect this.

Architecture & Software Interactions

- We heavily encourage referring to our GitHub Home Page for the full PNG files.
  - The PlantUML Format is also provided on the GitHub Home Page.

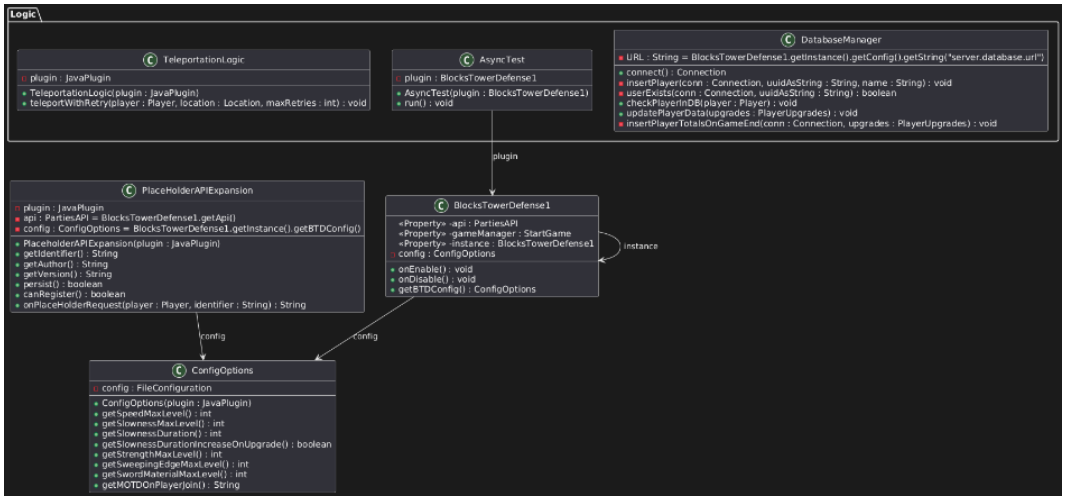

Figure 1: Logic_General: Handles overall game logic

Figure 2: Logic_Mob: Handles Mob interactions

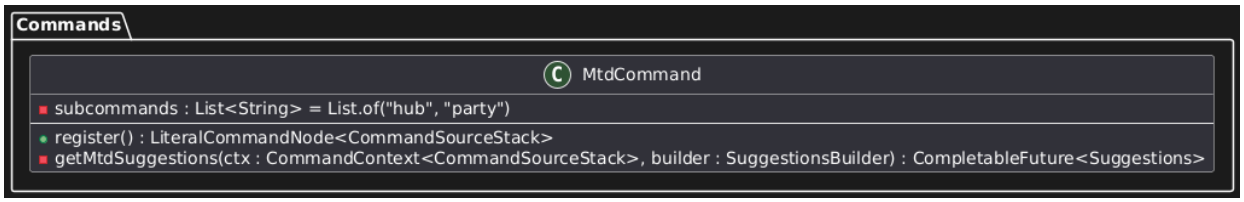

Figure 3: MtdCommand: Handles Player and Admin commands

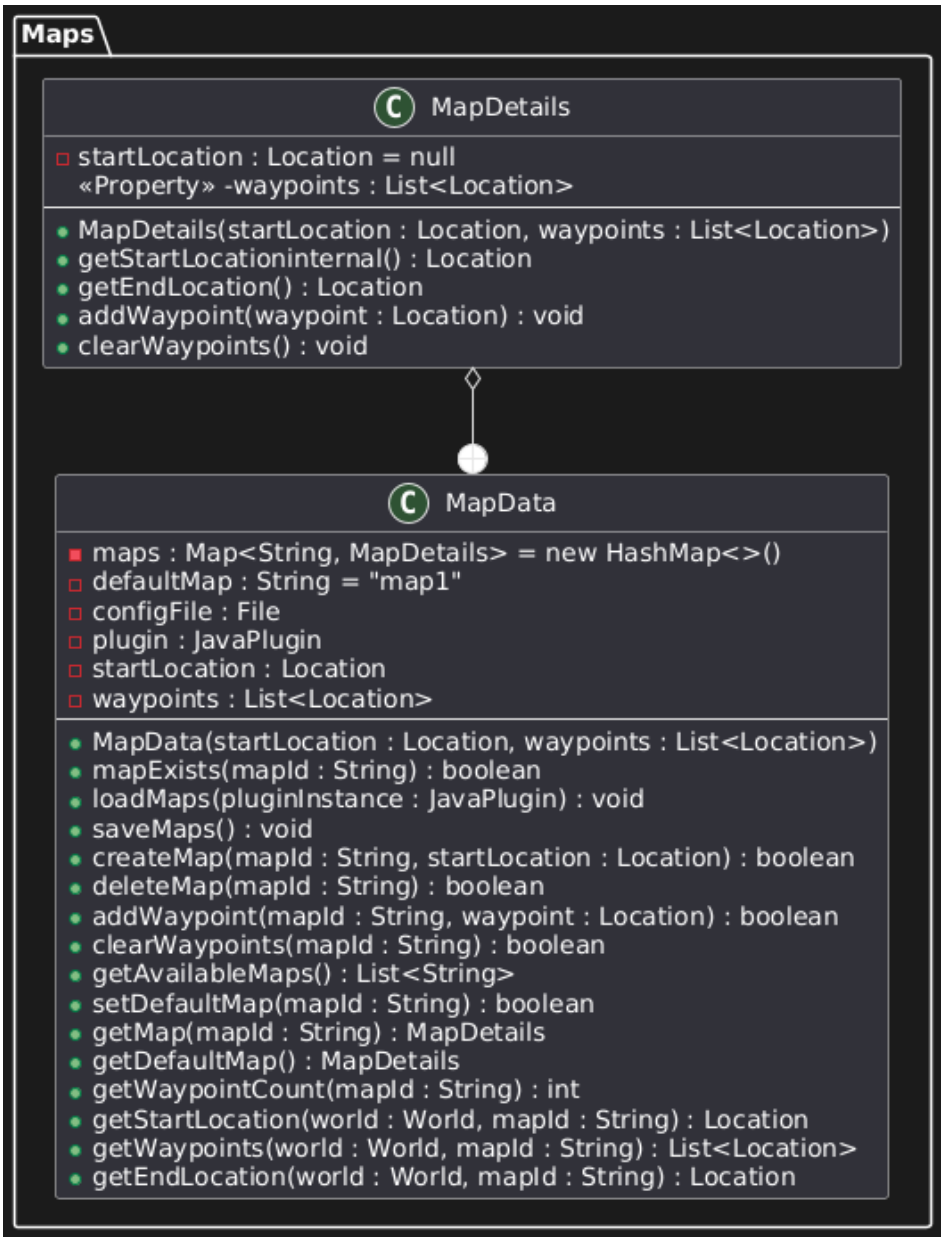

Figure 4: Maps: Handles Map Data & Mob Pathing

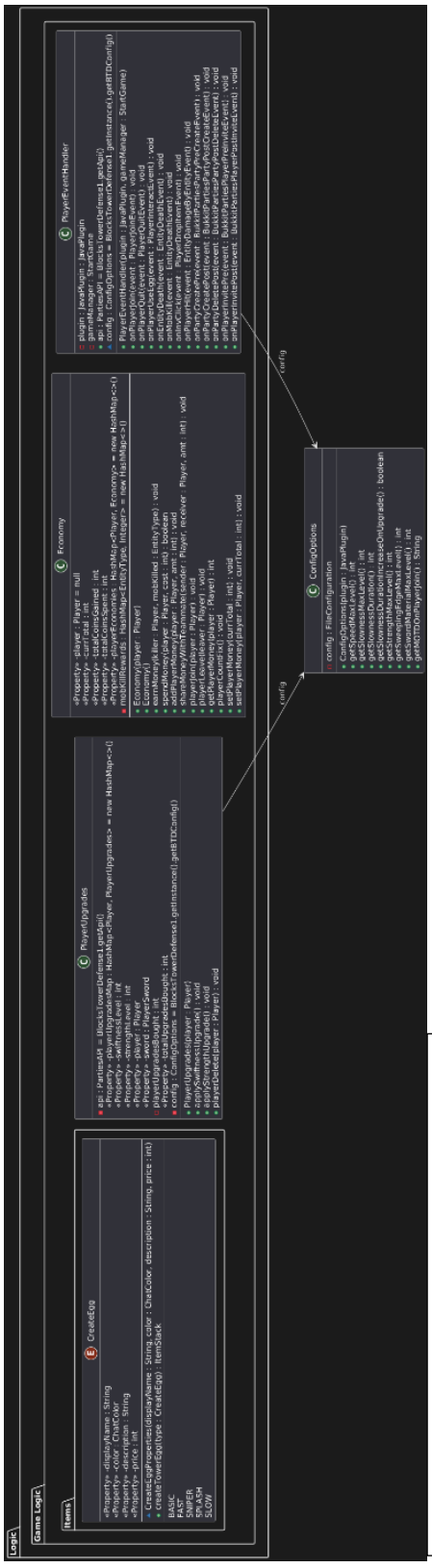

Figure 5: Logic_Player: Handles Player Interactions

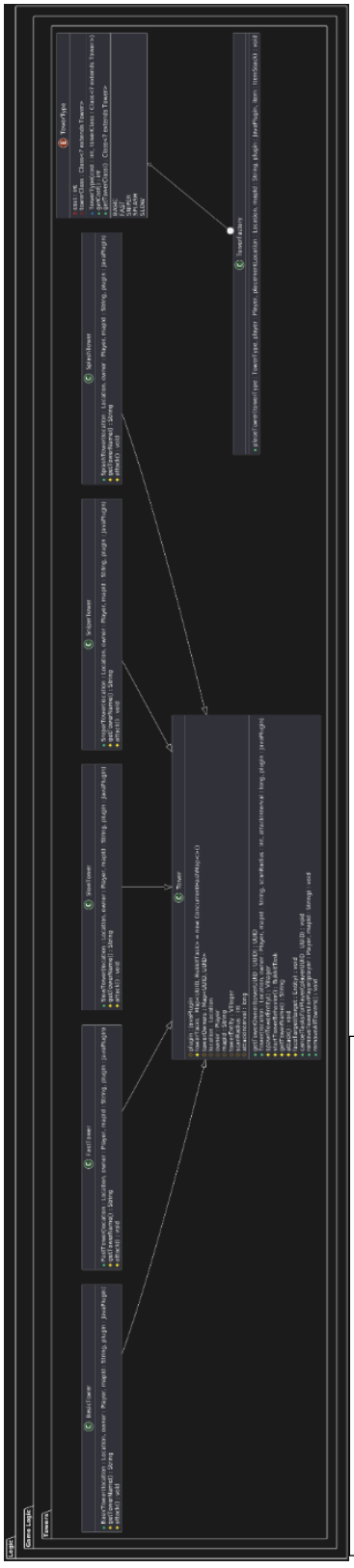

Figure 6: Logic_Tower: Handles Tower Interactions

Database Connection

- SQLite was utilized for our database connection as it is lightweight, free, and easy to implement
- Game data will be collected for user concurrency and data tracking to create the game's leaderboards system
    - Currently tracks the following: Gold Spent, Gold Gained, Games Played, Fastest Game Completion, and Total Upgrades Bought

Development

Planning & Game Design Document (GDD)

We started off with the Project Plan:

- A brief detailing of our timeline
    - Including class and sponsor tasks
- Planning the time spent on each part of the requirements
    - Reflected on our Gantt Chart
- General information regarding how we would be creating the game
- Weekly meetings with the sponsor
- Version control under a GitHub organization account

Then, we created a Game Design Document, essentially creating our project on paper. This document detailed:

- How the game mechanics would function
- Rough ideas for maps, mobs, players, and how the game would play out
- A list of commands and architecture behind the game
- The resources and constraints that would be utilized for our design

Gameplay & Mechanics

Initially, we settled upon building a game similar to Bloons Tower Defense but involving the player in combat like in Sanctum 2. This general idea was followed to the end, albeit with most changes on how the player would do combat and how we would handle towers. The general game rules and mechanics as seen in many tower defenses were followed:

- Players have limited hearts
- Hostile mobs would enter the map, follow the path, and either remove hearts should they reach the end or supply coins if defeated before then
- Players would build towers to combat these mobs
    - Tower upgrades would be more generalized and focus on styles
    - Player combat expanded with upgrades and consumables
- Players could set record scores to mark themselves on the leaderboard
- There would be many maps that players could play on with varying styles
    - These maps would have their own design philosophies and some having their own obstacles

Software Test Plan

How Will the Project Be Tested?

The project will be tested through actively playing the plugin and running through standard player interactions alongside potential unpredicted actions.

Who Will Test What Parts?

The project will be tested with all the members of the team based on what the topic entails and if a second member is needed, notably for multiplayer functionality.

What Aspects Need to Be Tested?

Refer to the Software Test Table, Column 1.

(Text) refers to the type of testing, with the text following being the specific test.

The known bugs on the GitHub Wiki will also be tested and marked in their corresponding test requirement. Said known bugs may be outdated and solved by the time of this report.

Where Will the Software Be Tested?

The software will currently be tested on a local server. The server will also be tested on the KSU MC Server upon finalization and implementation.

Software Test Report

Testing Table

Requirements	Pass	Fail	Severity
Plugin Installation			
(Configuration) Modifiable values			
(Configuration) Map creation			
Command functionality			
(Parties) Party creation			
(Parties) Income distribution			
Game mode startup			
(Gameplay) Player interactions			
(Gameplay) Tower interactions			
(Gameplay) Mob interactions			
(Gameplay) Round management			
Multiple ongoing games			
Map Creation			

Discussion

At the time of submission, April 20th, 2025, this report's testing has found no issues.

Version Control

For version control, we utilized GitHub and its branches to maintain our code changes. Our documentation would be saved within our old documents and reports, which would then be reformatted and stylized on the GitHub Wiki, akin to a game guide format. Maps would be saved on a local world and through WorldEdit and later also moved. Maps are to be saved to WorldEdit's schematics API which allows for file sharing and storing that can then be loaded in the server. All files worked on would also be moved onto the GitHub home page and uploaded as a PDF for accessibility.

Summary

Overall, as per the requirements of our Sponsor and through the reports for CS 4850, we feel that our project has been an incredible success. With IntelliJ, Paper, and various tools, we have met the requirements: Documentation and information necessary for its maintenance and is properly supplied within the GitHub page and the game is fully functional and customizable while having good performance, which we hope will stay this way once it is implemented on the KSU MC Server.

Appendix A – Supplementary Files

- Project Plan (January 26th)
- Game Design Document (February 10th)
- C-Day Poster (April 22nd)
- GitHub Wiki (Detailed Documentation)

Appendix B – Dependencies

- PartiesAPI Plugin
- PlaceholderAPI Plugin
- SimpleScore Plugin

Appendix C – Resources

- IntelliJ Community Edition
- IntelliJ Minecraft Development Plugin
- Spigot Development Guide
- Paper Development Guide
- GitHub
- PartiesAPI Plugin
- PlaceholderAPI Plugin
- WorldEdit Plugin
- SimpleScore Plugin
- PlantUML